Towards Software Verification for TinyOS Applications

Doina Bucur and Marta Kwiatkowska

Computing Laboratory, Oxford University, UK

{doina.bucur, marta.kwiatkowska}@comlab.ox.ac.uk

Introduction For the mainstream sensor operating system TinyOS, a programmer writes concurrent, shared-memory software in either nesC or the recent C *TosThreads* API [3]. Elusive *concurrency errors* arise because of the nondeterministic thread interleavings, while *context-awareness errors* are due to the application's inability to deal with unexpected context. We contribute the very first software verification tool for multithreaded, adaptive TinyOS 2.x applications written in TinyOS's C TosThreads API. We statically verify a TinyOS application against a safety specification w.r.t actuation and memory configuration, given a—possibly nondeterministic—pattern of incoming context data.

Our tool [1] builds on SATABS [2], a generic software verification tool for ANSI C; SATABS takes specifications written as user-specified assertions of boolean conditions inserted in the code. It supports complex program features such as dynamic thread creation, infinite loops and ANSI-C pointer use. We (i) add native support for the C TosThreads API to SATABS, (ii) implement a SATABS-readable C model of the TinyOS system calls to stand in for the OS kernel, and finally (iii) verify application and kernel model against context-aware safety specifications written as SATABS assertions. We report benchmarks on running our tool on standard applications distributed with TinyOS's sources, and on a more complex healthcare application; we find routine violations of safety requirements in staple TinyOS code.

The Tool A TinyOS application programmed in the TosThreads C API [3] is tightly connected to the rest of the operating system's kernel by calling kernel services; these either manage execution scheduling (e.g. thread creation and suspending) or access hardware (e.g. the radio port, sensor chips or resident leds) on their respective software interfaces. We replace these services with a kernel *model*, ensuring that their interface behaviour is preserved; e.g., if `amRadioSend([..])` can fail returning `EBUSY`, so must its model.

We give the results of our verification runs in Table 1; the Lines of Code (LOC) count includes the necessary segments of kernel model. *SenseAndSend* is the staple monitoring application in the TinyOS source tree. Four working threads monitor the four Tmote Sky on-board sensor chips, and each writes a fresh value in the data field of a network message; a fifth thread sends the message on the radio. Claim 79 uncovers a misuse of the radio interface; thread 0 fails to ensure that the radio is turned on by not checking `amRadioStart`'s returned error code, before a call to `amRadioSend`:

Table 1: Verification benchmarks. Times are given for runs on a Mac OS X with a 2.4GHz Duo Intel Core and 2GB RAM.

Application (Threads/LOC)	**Claim line**	**Verified?**	**Time**	**Error: context awareness**	**Error: concurrency**
Blink 4/64	66	yes	2.9s	-	-
SenseAndSend 6/347	79	**no**	32.2s	interface use	order violation
	136	**no**	1m08s	sensing failure	-
	146	yes	4m25s	-	-
PatientNode 6/439	230	**no**	35m07s	network failure	deadlock
	268	yes	2m38s	(false reasoning)	-
	262	yes	61m12s	-	-

```
amRadioStart();                                // thread 0, main
if(amRadioSend(AM_BROADCAST_ADDR, &send_msg, ..) // thread 5, sending
```

The overall specification (claim 146) states that such messages should be sent periodically, containing valid readings, and accompanied by led signalling.

PatientNode is a SenseAndSend extension for monitoring patients in a health-care network. A number of biosensors monitor each patient; a PatientNode application resident on one such sensor collects readings from the patient's sensors, sends them in a network message, and signals an abnormal condition by a lit-led configuration. Claim 230 uncovers that a misplaced closing brace brings the program into a deadlock on a barrier, if a message expected to be received doesn't show up:

```
if(amRadioReceive(&recv_msg, [..]) == SUCCESS) { // thread 3, receiving
  barrier_block(&send_barrier);
}
barrier_block(&send_barrier);                  // thread 5, sending
barrier_reset(&send_barrier, [..]);
amRadioSend(AM_BROADCAST_ADDR, &send_msg, [..]);
```

Claim 268 verifies application logic: an abnormal received reading must be treated as a false alarm if it is not confirmed by a subsequent such reading.

Acknowledgments The authors are supported by the project *UbiVal: Fundamental Approaches to Validation of Ubiquitous Computing Applications and Infrastructures*, EPSRC grant EP/D076625/2.

References

[1] Doina Bucur and Marta Kwiatkowska. Bug-Free Sensors: The Automatic Verification of Context-Aware TinyOS Applications. In *Proceedings of the Third European Conference on Ambient Intelligence (AmI)*. Springer, 2009.

[2] Edmund Clarke, Daniel Kroening, Natasha Sharygina, and Karen Yorav. SATABS: SAT-based Predicate Abstraction for ANSI-C. In *TACAS*, 2005.

[3] Kevin Klues, Chieh-Jan Liang, Jeongyeup Paek, Răzvan Musăloiu, Ramesh Govindan, Andreas Terzis, and Philip Levis. TOSThreads: Safe and Non-Invasive Preemption in TinyOS. In *ACM SenSys*, 2009.